Python

13

Методы класса

```
class Car:
    __last_model = None
    def __init__(self, model):
        self.model = model
        Car.__last_model = model
    @classmethod
    def get_last_model(cls):
        return cls.__last_model
car1 = Car('A')
print(Car.get_last_model())
```

Методы класса - методы которые доступны напрямую из класса.

Методы экземпляров - требуют обращения через экземпляр класса.

Методы класса - позволяют обращаться как через экземпляр класса, так и через сам класс. Могут оперировать только атрибутами класса.

Задание 13.01

Создать метод класса get_counter. Создать три объекта класса. Вызвать через класс метод get_counter.

Статические методы

```
class Car:
    __last_model = None

    def __init__(self, model):
        self.model = model
        Car.__last_model = model

    @staticmethod
    def is_model_ok(model):
        return len(model) > 3:

print(Car.is_model_ok('abc'))
```

Статичный метод - метод определенный внутри класса, и не имеющий доступа ни к методам класса, ни к атрибутам класса. Класс выступает хранилищем метода(“функции”).

Задание 13.02

Создать статичный метод get_random_name для класса Pet. Метод возвращает случайную строку вида A-42.

Множественное наследование

```
class A:
    def print_smth(self):
        print('a')
    def a_method(self):
        print('a method')

class B(A):
    def print_smth(self):
        print('B')
    def b_method(self):
        print('b method')
```

```
class C(A):
    def print_smth(self):
        print('C')

    def c_method(self):
        print('c method')

class D(B, C):
    def d_method(self):
        print('d method')
```

Задание 13.03

Унаследовать от класса Pet два класса: Horse, Donkey. Переопределить в классах методы voice.

Создать класс Mule. Метод voice должен быть унаследован от класса Donkey

Exceptions

```
a = int(input('a: '))
b = int(input('b: '))
try:
    result = a / b
except ZeroDivisionError as err:
    print(f'b is zero - {err}!!!')
except Exception as err:
    print(f'SOMETHING WRONG - {err}!!!')
```

Для ловли ошибок используется синтаксическая конструкция try..except. В блок try помещается участок кода, в котором ожидается некорректное поведение. В блоке except находится инструкции, которые применяются в случае появление ошибки

Exceptions

```
a = int(input('a: '))
b = int(input('b: '))
try:
    result = a / b
except ZeroDivisionError as err:
    print(f'b is zero - {err}!!!')
except Exception as err:
    print(f'SOMETHING WRONG - {err}!!!')
else:
    print('Ошибки не было')
finally:
    print('Сработает всегда')
```

В конец блока try..except можно добавить еще два блока else и finally.

Блок else выполнится только в том случае, если блок try выполняется без ошибки.

Блок finally выполняется всегда, даже в том случае если в блоке except была вызвана другая, необработанная ошибка

Вызов ошибки

```
a = int(input('a: '))
b = int(input('b: '))

if b == 0:
    raise ZeroDivisionError('b is zero')
```

Создание собственных ошибок

```
class MyException(Exception):
    def __init__(self, message='AAA!!'):
        super().__init__(message)


raise MyException
```

Задание 13.04

Создать класс Book. Атрибуты: количество страниц, год издания, автор, цена. Добавить валидацию в конструкторе на ввод корректных данных. Создать иерархию ошибок.

Абстрактный класс

```
from abc import ABC, abstractmethod

class A(ABC):
    @abstractmethod
    def do_smth(self):
        print('I am a parent')
class B(A):
    def do_smth(self):
        print('I am a child')
a = A()  # ERROR
b = B()
```

Абстрактный класс - класс, экземпляр которого нельзя создать.

Задание 13.05

Сделать класс Pet абстрактным

Интерфейсы

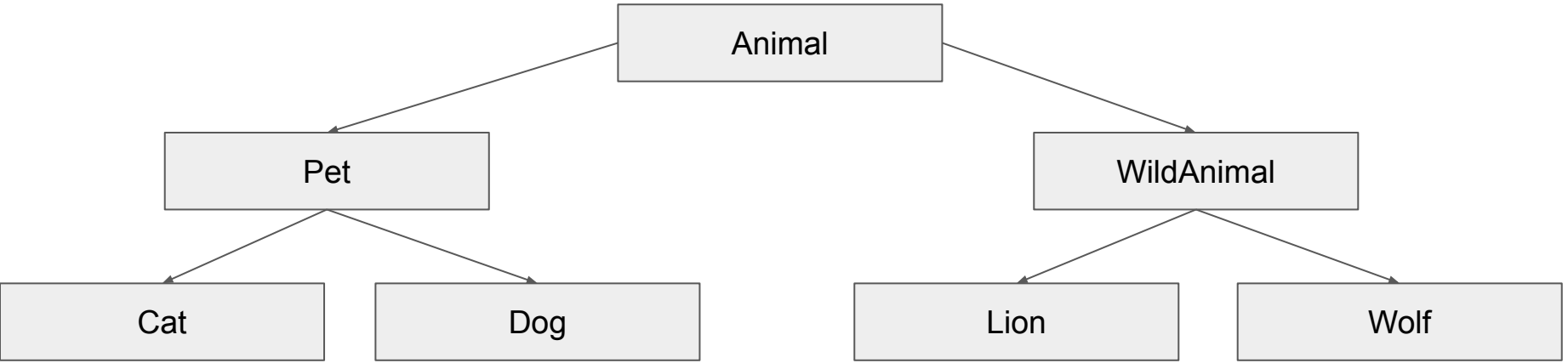
```
class MyInterface(ABC):
    @abstractmethod
    def do_a(self, arg1):
        raise NotImplemented
    @abstractmethod
    def do_b(self, arg1, arg2):
        raise NotImplemented

class MyClass(MyInterface):
    def do_a(self, arg1):
        print(arg1)
    def do_b(self, arg1, arg2):
        print(arg1, arg2)
```

Интерфейсы - класс, определяющий методы и их сигнатуру, который **Должны** быть переопределены в дочерних классах. Интерфейс - договор

Задание 13.06

Реализовать следующую структуру:



Задание 13.07

Реализовать интерфейсы: Feline(), Canine()

Mixins

```
class MyMixin:
    def do_a(self):
        print(self.a)
    def do_b(self, arg1, arg2):
        print(self.b)

class MyClass(MyMixin):
    def __init__(self, a, b):
        self.a = a
        self.b = b
```

Миксины инкапсулируют поведение которое может быть использовано в классах.

Модули

Модуль - файл с расширением .py

Модули содержат классы, функции, константы

Импортирование модулей

```
import datetime

from datetime import datetime as da
```

Пакеты

Пакет - католог, в котором находятся другие каталоги и/или модули и содержащий файл __init__.py.

Задание

Создать пакет следующей структуры:

src/

-matrix_utils/

--matrix_classes.py

--matrix_funcs.py

-main.py

Задание 13.07

Создать класс Matrix. Атрибуты - data(содержит саму матрицу - список списков), n, m. Определить конструктор(с параметрами(передача размерности: n, m и диапазона случайных чисел: a, b), по-умолчанию (матрица 5 на 5 где все элементы равны нулю), копирования) , переопределить магический метод __str__ для красивого вывода.

Описать функции, которые принимают на вход объект класса Matrix. Функции позволяют искать максимальный элемент матрицы, минимальный, сумму всех элементов.

Создать в файле main.py матрицу. Воспользоваться всеми описанными функциями и методами